



# Workflows, Teams, and Integrations



## Who We Are!



- Rob, Nathan, Aurora, Richard
- Members of the Dev Team at Pretio Interactive
- Support multiple code bases in multiple languages with multiple deploys and builds
- Suck at mind reading, and thus rely on teamwork and communication



## The Plan:

- **High Level Concepts**
  - Why this matters
  - Different git branch models
  - How Testing and Code Review fit
  - Continuous Deployment
  - Scheduled Deployment
- **Nitty Gritty**
  - Commit messages
  - log, blame, reflog etc.
  - Rebase vs Merge
  - Grab-bag of useful commands
- **Interactive Workshop**
  - we all commit to the same repository, fun happens!



## A Note About GUIs...

- Git GUI Tools are out there and are cool!
  - <http://www.git-tower.com/>
  - <https://desktop.github.com/>
- IDEs sometimes have nice Git integration
  - <https://www.jetbrains.com/>
- Today we will focus on the command line



- GUI tools good for designers, artists, folks who work with assets and don't care about code
- Still need a command-line Git Master, it might as well be you

# Quick Refresher

- **Clone a project from a server (GitHub example):**
  - `git clone git@github.com:Pretio/startup_slam_2015_website.git`
- **See your Remote Info:**
  - `git remote -v`
- **See your Branches & Info:**
  - `git branch -vv`
- **Switch Branches:**
  - `git checkout master`
- **Create a new Branch:**
  - `git checkout -b TASK-123 --track origin/master`
- **Check the current 'state' of your local branch against your origin:**
  - `git status`
- **Commit changes to your local branch**
  - `git add .`
  - `git commit -m "Task-123: adds ability to recover your password via an email submit form"`
- **Pull updates from origin (on the branch you are tracking):**
  - `git pull --rebase`
- **Merge content (locally) from another branch to the branch you are currently on:**
  - `git merge production`
- **Push content from your local branch to a branch on the server:**
- **(GitHub)** `git push origin TASK-123:TASK-123`
- **(Gerrit)** `git push origin HEAD:refs/for/master`



- Don't forget to mention at the beginning of this that GitHub != Git!

## Whyfor?

- Keeps history in a maintainable way
- Communicates changes to the rest of the team
- Smallest possible Dev Team is 3



## Configuration Tools

- Configure your local git install for fun and profit
- git-completion.bash
  - <https://git-scm.com/book/en/v1/Git-Basics-Tips-and-Tricks>
- git-prompt.sh
  - <http://git-prompt.sh/>
  - <http://code-worrier.com/blog/git-branch-in-bash-prompt/>
- Talk to other Dev's, folks you work with etc.



- git-completion.bash (tab auto-complete for commands, branch names etc)
- git-prompt.sh (shows you which branch you are on in your prompt)

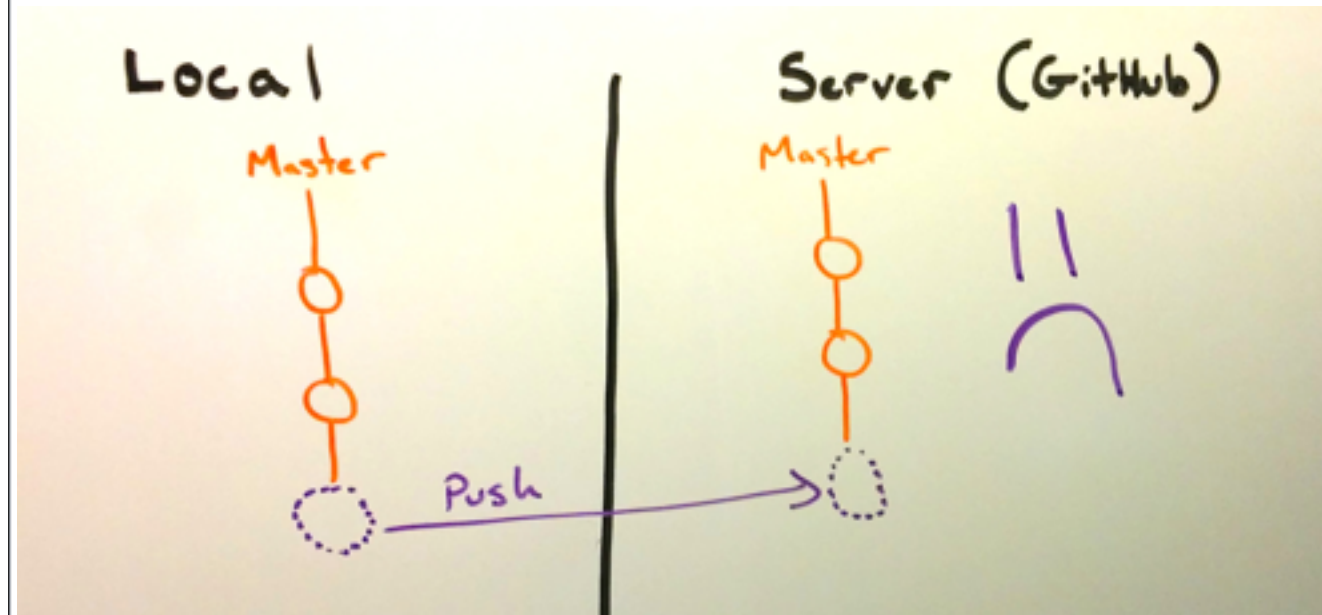
# Multi-Branch Development



- if you are working off of a single branch called 'master' then stop that now
- branches are disposable!
- use feature branches on your local

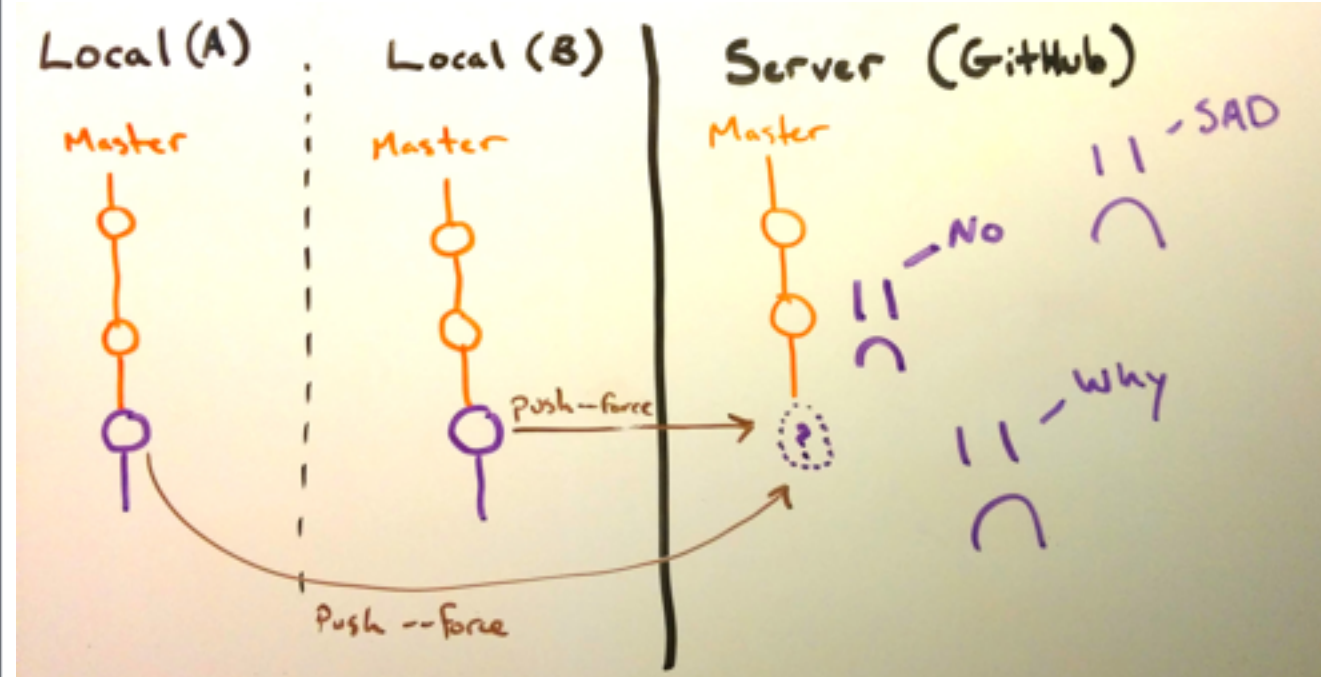


## Single-Branch Example



- This is the conceptual model of the previous slide, get used to this drawing notation

## Single-Branch Example with Two People



## Multi-Branch Development

- Automation is your friend!
- Use Continuous Integration Software to:
  - Automatically Run Tests
  - Auto-Merge code from one branch to another
  - Produce a production 'build' of our software and put it on the production servers
- 'Webhooks' watch your GitHub repo and trigger events in response to GitHub events:
  - When a pull request is made
  - When a merge occurs
- We use the free 'Jenkins' (<http://jenkins-ci.org/>) but there are others out there



- alternatives include 'Bambo' and 'TeamCity'

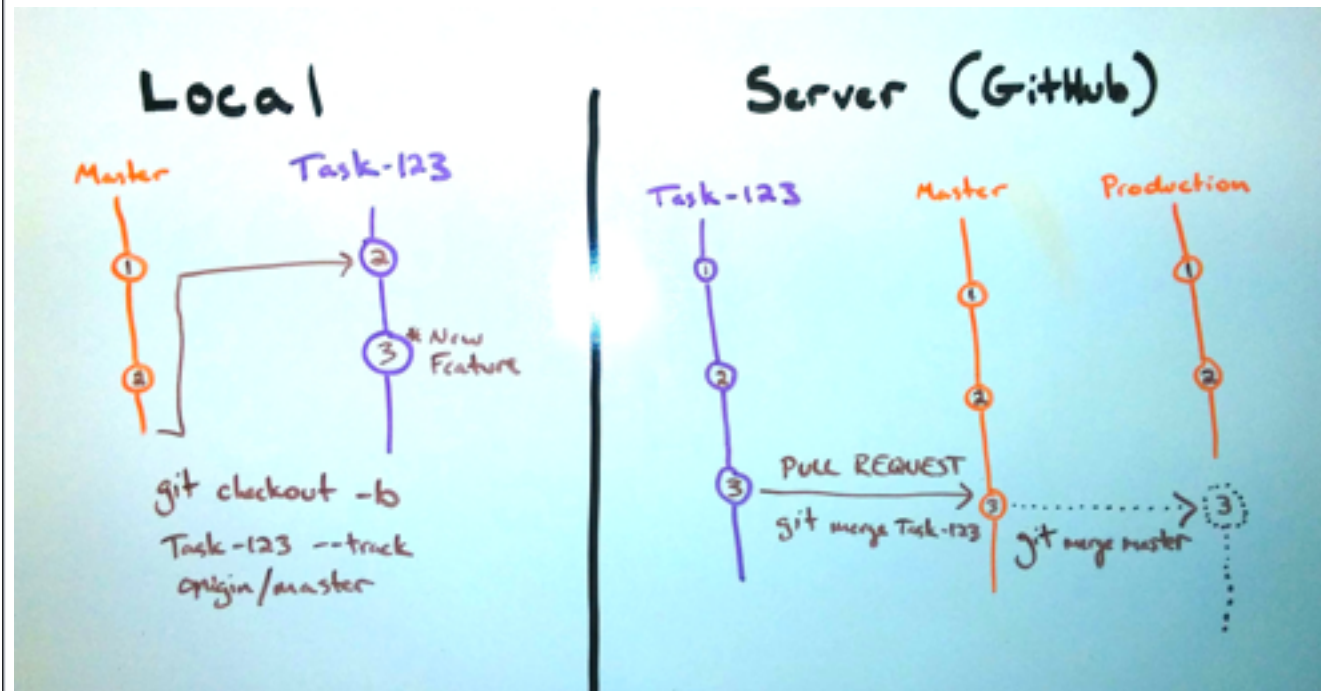
# Multi-Branch Development

The screenshot shows the GitHub interface for the repository 'pretio / pretio-api'. The 'All branches' tab is selected, displaying a list of branches. The 'master' and 'production' branches are locked, indicated by a padlock icon. Other branches like 'add\_profiling', 'restore\_error\_limit\_hits', 'fix\_for\_redempt ion\_conflicts', and 'revert-398-cacheable\_credentials' have associated pull requests. Below the screenshot, a terminal window shows the command 'git branch' being executed, resulting in a list of branches: PRET-262, PRET-287, PRET-405, db\_replica\_session, \* master, production, and redis\_sorted\_set.

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git branch
PRET-262
PRET-287
PRET-405
db_replica_session
* master
production
redis_sorted_set
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$
```

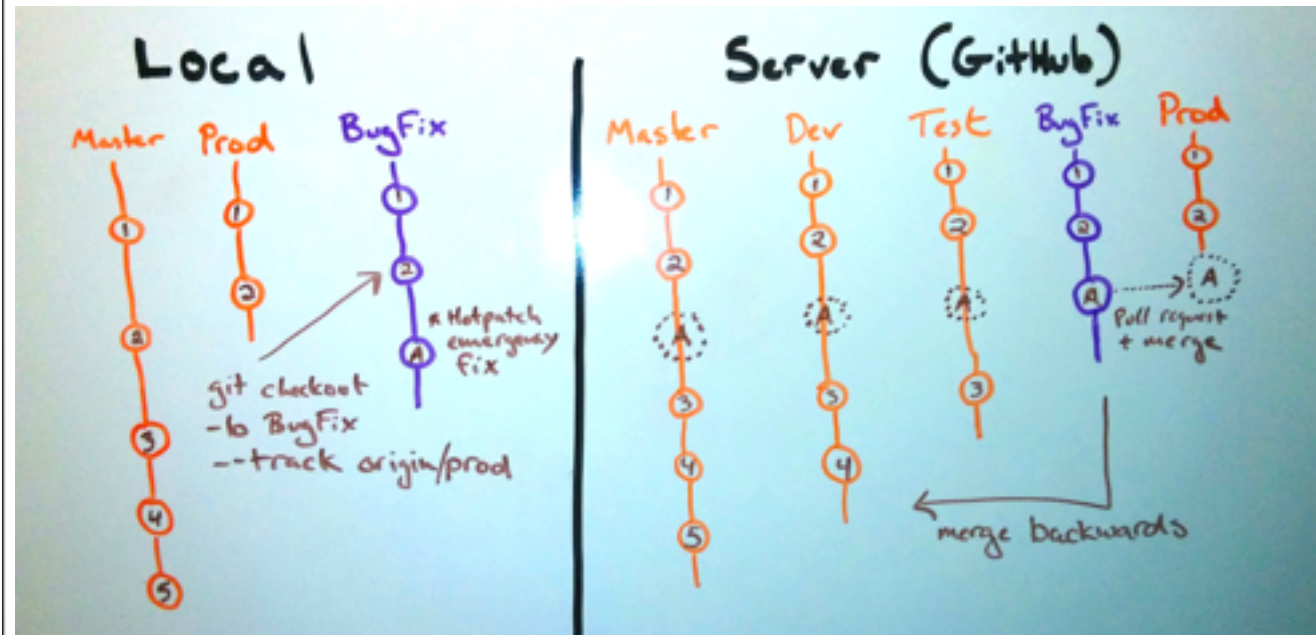
- note locked 'master' and 'production' branches (show new github feature)
- rest are feature branches actively being worked on, will be thrown away when done

## Multi-Branch Example: Continuous Deploy



- Example of a Continuous Deploy model - light, code on Prod auto-rolls out
- orange are your 'protected' branches
- purple the feature (disposable) branch

## Multi-Branch Example: Scheduled Deploy



- Scheduled builds, slower control to merge forward
- orange 'protected' branches, purple bug fix tracking production
- Dev/Test branches used to make interim builds, i.e. to an internal test server
- Note this merge is a hint for the rebase vs merge slide

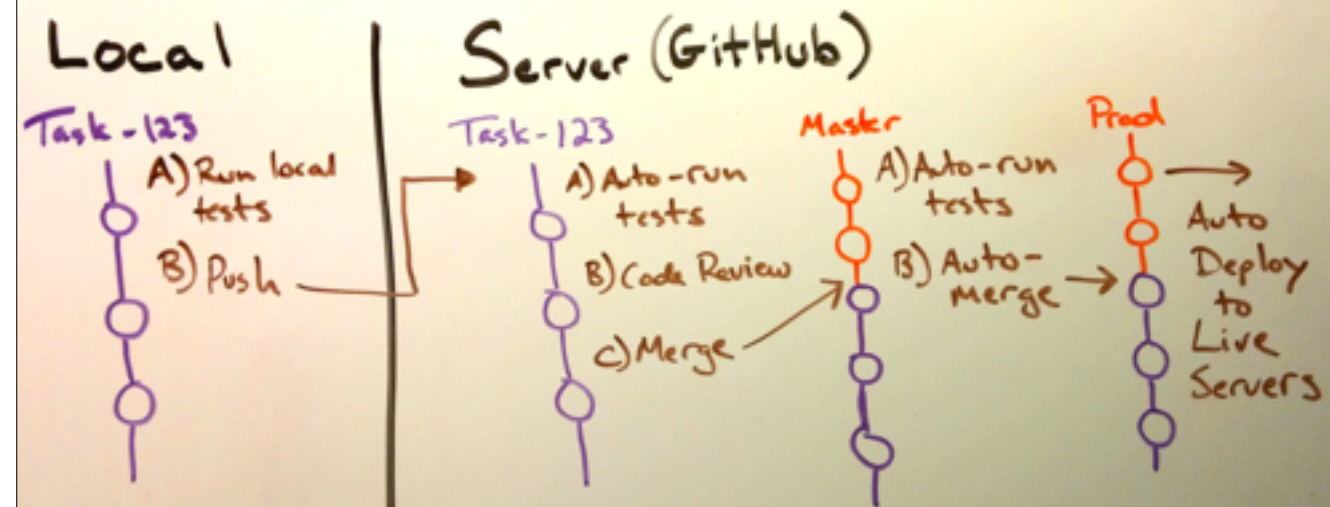
## Code Review & Tests

- Code Review happens between your feature branch and master
  - Could be via a 'Pull Request' as on GitHub
  - Could be via tools such as 'Gerrit'
- Tests should be run often!
  - Should be run manually locally
  - Should be run before Code Review
  - Should be run before merge to 'production' branch



- Review a) helps prevent errors b) increases team knowledge of code, c) teaches new members code conventions
- Tests are critical, even for one liner hot fix changes on prod

# Code Review & Tests in Flow





And unless you have some questions on that previous bit...

## **NOW ON TO THE NITTY GRITTY DETAILS**



## Play Along at Home!

Feel free to checkout the SendWithUs Startup Slam project:

```
> git clone git@github.com:Pretio/startupslam.git
```

Crack open a Terminal Window, and play along locally trying out the commands!



## Good Commit Messages

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT  
MESSAGES GET LESS AND LESS INFORMATIVE.

## Good Commit Messages: Why

- commit messages are **passive** forms of communication with all temporal versions of you and the members of your team
- best case scenario when you write a bad commit message is that it is ignored
- in an emergency situation (say a rollback) no one wants to read the code associated with a commit



## Good Commit Messages: How

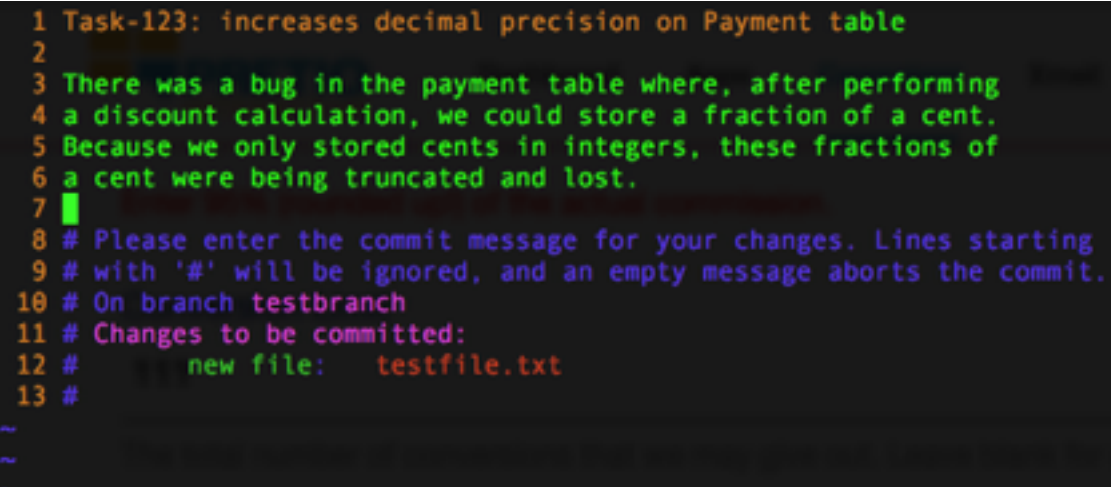
- limit the subject to 50 characters (note word-wrap on GitHub)
- if needed, add a paragraph description (using *git commit* instead of *git commit -m*)
- commit should explain ‘what’ and ‘why’
- the code explains ‘how’
- use the imperative mood: “This commit will....”



- see in github how words wrap funny - that's the character limit

# Good Commit Messages: How

```
> git commit
```



```
1 Task-123: increases decimal precision on Payment table
2
3 There was a bug in the payment table where, after performing
4 a discount calculation, we could store a fraction of a cent.
5 Because we only stored cents in integers, these fractions of
6 a cent were being truncated and lost.
7
8 # Please enter the commit message for your changes. Lines starting
9 # with '#' will be ignored, and an empty message aborts the commit.
10 # On branch testbranch
11 # Changes to be committed:
12 #   new file:   testfile.txt
13 #
```

```
> git commit -m "Insert Message Here"
```



```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git commit -m "PRET-435: increases decimal precision on Payment table"
```

- first command opens the commit in your command line editor (Vim is pictured)
- second command is a shortcut for a quick message



## Git logs and history: Why

- Lots of git tools that tell you what happened, when, and (if the person wrote good commit messages!) why
- Mission critical when you are working on a code base written by someone else
- Mission critical when you are working on a code base written by past you, and present you suspects past you was drunk



## Git logs and history: How

> git log

```
bash      pretio@worker:~/repo
commit 830518ae93ad1f56135af6498e006f34edb8c932
Author: Gregory Schier <gschier1990@gmail.com>
Date:   Fri Sep 18 12:29:31 2015 -0700

    Some tweaks

commit 3127417d3f0656ec5088d6194399228bf7b51268
Author: Noah Warder <nwarder@users.noreply.github.com>
Date:   Fri Sep 18 12:22:52 2015 -0700

    Update index.html

commit 1164461099f3a9ebcb35e206440c9456e4df855e
Author: Brad Van Vugt <bvanvugt@gmail.com>
Date:   Thu Sep 17 13:08:06 2015 -0700

    Add placeholders for workshop resources.

commit d94d7eb172176f1941c4364977ed73259644c538
Author: Gregory Schier <gschier1990@gmail.com>
Date:   Thu Sep 17 12:06:38 2015 -0700

    Silkstart Logo

commit 5879d3099507723030cb7a3453943edd27bac545
Author: Brad Van Vugt <bvanvugt@gmail.com>
Date:   Sun Sep 13 00:01:33 2015 -0700

    Fix schedule ordering.
```

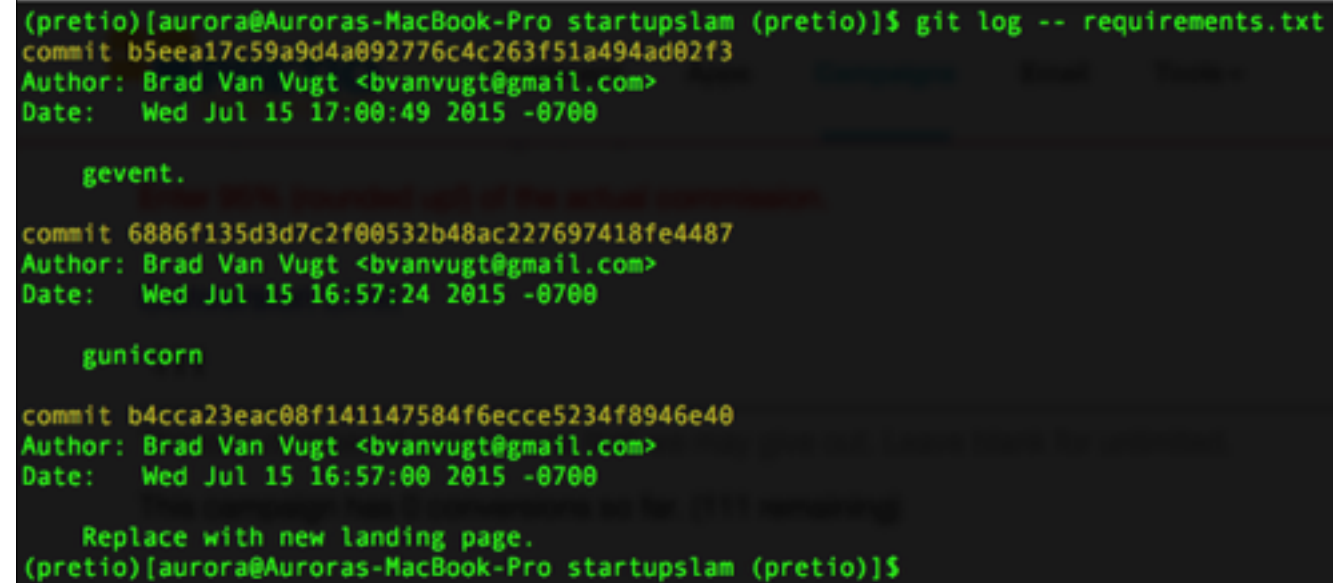


- standard historical log in reverse chronological order
- this log is taken from the startupslam git repo. I'll leave judgement of the quality of the log to the reader.
- NOTE THE HASH OF THE COMMIT (used for other commands)



## Git logs and history: How

```
> git log -- <filename>
```

A terminal window screenshot showing the output of the command 'git log -- requirements.txt'. The output lists three commits by Brad Van Vugt. The first commit (b5eea17c59a9d4a092776c4c263f51a494ad02f3) is dated Wed Jul 15 17:00:49 2015 and includes the message 'gevent.'. The second commit (6886f135d3d7c2f00532b48ac227697418fe4487) is dated Wed Jul 15 16:57:24 2015 and includes the message 'unicorn'. The third commit (b4cca23eac08f141147584f6ecce5234f8946e40) is dated Wed Jul 15 16:57:00 2015 and includes the message 'Replace with new landing page.'. The terminal prompt is '(pretio)[aurora@Auroras-MacBook-Pro startupslam (pretio)]\$'.

```
(pretio)[aurora@Auroras-MacBook-Pro startupslam (pretio)]$ git log -- requirements.txt
commit b5eea17c59a9d4a092776c4c263f51a494ad02f3
Author: Brad Van Vugt <bvanvugt@gmail.com>
Date:   Wed Jul 15 17:00:49 2015 -0700

    gevent.

commit 6886f135d3d7c2f00532b48ac227697418fe4487
Author: Brad Van Vugt <bvanvugt@gmail.com>
Date:   Wed Jul 15 16:57:24 2015 -0700

    unicorn

commit b4cca23eac08f141147584f6ecce5234f8946e40
Author: Brad Van Vugt <bvanvugt@gmail.com>
Date:   Wed Jul 15 16:57:00 2015 -0700

    Replace with new landing page.
(pretio)[aurora@Auroras-MacBook-Pro startupslam (pretio)]$
```

- specify by filename for just the history on that file/folder (works ESPECIALLY great on deleted files, and trying to figure out when a file got deleted)
- Note equivalent PyCharm hack: right-click on file, git -> history

## Git logs and history: How

```
> git blame -- <filename>
```

```
(pretio)[aurora@Auroras-MacBook-Pro startupslam (pretio)]$ git blame -- app.py
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 1) import bottle
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 2)
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 3)
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 4) @bottle.get('/')
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 5) def index():
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 6)     return bottle.static_file('index.html', root='')
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 7)
8997fbc3 (Dylan Moore 2015-08-23 11:00:33 -0700 8) @bottle.get('/static/<filepath:path>')
8997fbc3 (Dylan Moore 2015-08-23 11:00:33 -0700 9) def server_static(filepath):
8997fbc3 (Dylan Moore 2015-08-23 11:00:33 -0700 10)     return bottle.static_file(filepath, root='static')
8997fbc3 (Dylan Moore 2015-08-23 11:00:33 -0700 11)
b4cca23e (Brad Van Vugt 2015-07-15 16:57:00 -0700 12) application = bottle.default_app()
```



- Show who changed exactly what line in what commit (shortened hash, but usable most anywhere you can use a commit)

# Git logs and history: How

```
> git show <commit>
```

```
commit 8e3044be192843e235ef4a11a1faecb272012b5a
Author: Aurora Walker <aurora.walker@pretiointeractive.com>
Date:   Wed Mar 4 13:39:49 2015 -0800

    Updating deploy scripts and ansible playbook for rolling deploys

diff --git a/.gitignore b/.gitignore
index 7c84665..8d70b2d 100644
--- a/.gitignore
+++ b/.gitignore
@@ -56,5 +56,8 @@ node_modules
 bower_components
 .module-cache
+
+#Mac specific files
+.DS_Store
+
+# Pycharm specific project files
+.idea/
diff --git a/deploy/app.yml b/deploy/app.yml
index d5a5e29..cd1e3c2 100644
--- a/deploy/app.yml
+++ b/deploy/app.yml
@@ -17,7 +17,8 @@
- api
```



- Full diff of a commit with all changes
- minus sign means a removal, plus sign means an addition
- practically speaking the file dif on GitHub is much more user friendly

## Git logs and history: How

```
> git show --name-only <commit>
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git show --name-only 8e3044be192843e235ef4a11a1faecb272012b5a
commit 8e3044be192843e235ef4a11a1faecb272012b5a
Author: Aurora Walker <aurora.walker@pretiointeractive.com>
Date:   Wed Mar 4 13:39:49 2015 -0800

    Updating deploy scripts and ansible playbook for rolling deploys

.gitignore
deploy/app.yml
deploy/ec2.yml
deploy/group_vars/all
deploy/requirements.txt
deploy/roles/api/tasks/main.yml
deploy/roles/celery-base/tasks/main.yml
deploy/roles/webserver/tasks/main.yml
deploy/roles/webserver/templates/conf/nginx.conf
deploy/roles/webserver/templates/init/gunicorn.conf
deploy/roles/webserver/templates/init/nginx.conf
deploy/roles/webserver/vars/main.yml
deploy/scripts/provision
deploy/staging.yml
fabfile/deploy.py
fabfile/eb.py
fabfile/server.py
fabfile/utils.py
requirements.txt
```

- Show all files changed in a given commit
  - Disclaimer: 8e3044be192843e235ef4a11a1faecb272012b5a I use PyCharm to tell me these things rather than use the command line
- DEMO IN PYCHARM HOW TO DO THIS EASIER

## Git logs and history: How

```
> git revert <commit>  
//creates an 'opposite' commit that un-do's
```

```
commit ae910895f6fdcf02bc96670416c9d412c9ef5082  
Author: Aurora Walker <aurora.walker@pretiointeractive.com>  
Date:   Wed Sep 23 15:43:58 2015 -0700  
  
    Revert "PRET-435: adds custom webhook for Voleo and MailChimp"  
  
    This reverts commit 2e9a1af4636bb5970e8763fa02aff1a696650cb6.  
  
commit 2e9a1af4636bb5970e8763fa02aff1a696650cb6  
Author: Aurora Walker <aurora.walker@pretiointeractive.com>  
Date:   Tue Sep 22 09:23:55 2015 -0700  
  
    PRET-435: adds custom webhook for Voleo and MailChimp  
  
commit e558475d2c3c1fa39c93b608b474abd481e4df41  
Merge: b24d849 71247e0  
Author: Carl Stubens <carl@stubens.ca>  
Date:   Wed Sep 23 12:09:28 2015 -0700  
  
    Merge pull request #445 from Pretio/PRET-396-funnel-summary
```



## Git logs and history: How

```
> git reflog
// different from log! 'Personal' history
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git reflog
ae91089 HEAD@{0}: revert: Revert "PRET-435: adds custom webhook for Voleo and MailChimp"
2e9a1af HEAD@{1}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
9d09113 HEAD@{2}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
990e791 HEAD@{3}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
2ee5f95 HEAD@{4}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
d4d7652 HEAD@{5}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
9adf584 HEAD@{6}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
271de52 HEAD@{7}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
33b4cdd HEAD@{8}: rebase finished: returning to refs/heads/PRET-435
33b4cdd HEAD@{9}: pull --rebase: PRET-435: adds custom webhook for Voleo and MailChimp
e558475 HEAD@{10}: pull --rebase: checkout e558475d2c3c1fa39c93b608b474abd481e4df41
d7bff4c HEAD@{11}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
a902041 HEAD@{12}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
e259fb4 HEAD@{13}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
9c0d24f HEAD@{14}: rebase finished: returning to refs/heads/PRET-435
9c0d24f HEAD@{15}: pull --rebase: PRET-435: adds custom webhook for Voleo and MailChimp
b24d849 HEAD@{16}: pull --rebase: checkout b24d849df1bc4202d67ff6854004189be9a97195
948ead2 HEAD@{17}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp
e71138b HEAD@{18}: checkout: moving from master to PRET-435
c6f8cf3 HEAD@{19}: checkout: moving from PRET-435 to master
e71138b HEAD@{20}: checkout: moving from master to PRET-435
c6f8cf3 HEAD@{21}: checkout: moving from testbranch to master
c6f8cf3 HEAD@{22}: checkout: moving from master to testbranch
c6f8cf3 HEAD@{23}: checkout: moving from testbranch to master
c6f8cf3 HEAD@{24}: checkout: moving from master to testbranch
c6f8cf3 HEAD@{25}: rebase finished: returning to refs/heads/master
c6f8cf3 HEAD@{26}: pull --rebase: checkout c6f8cf3ea76443bfd6f5ad8b61a0c599e7c0fb6
```



- more like a history of your actions (across all branches)
- allows you to move back and forth within your personal timeline

## Git logs and history: How

```
> git reset HEAD@{#}  
// go to a specific state in your reflog (local)
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git reset HEAD@{9}  
Unstaged changes after reset:  
M    pretio/config/env.py  
M    pretio/database/webhooks/voleo_mailchimp.py  
M    test/database/webhooks/voleo_mailchimp_test.py  
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git reflog  
33b4cdd HEAD@{0}: reset: moving to HEAD@{9}  
2e9a1af HEAD@{1}: reset: moving to HEAD^  
ae91089 HEAD@{2}: revert: Revert "PRET-435: adds custom webhook for Voleo and MailChimp"  
2e9a1af HEAD@{3}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
9d09113 HEAD@{4}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
990e791 HEAD@{5}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
2ee5f95 HEAD@{6}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
d4d7652 HEAD@{7}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
9adf584 HEAD@{8}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
271de52 HEAD@{9}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
33b4cdd HEAD@{10}: rebase finished: returning to refs/heads/PRET-435  
33b4cdd HEAD@{11}: pull --rebase: PRET-435: adds custom webhook for Voleo and MailChimp
```



- Go to a specific point in time, but preserves time
- Note that it bumps everything up in the list!
- This is the master tool of fixing shit



## Git logs and history: How

```
> git reset --soft HEAD^  
// --soft leaves changes in 'status'
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git reset --soft HEAD^  
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git reflog  
2e9a1af HEAD@{0}: reset: moving to HEAD^  
ae91089 HEAD@{1}: revert: Revert "PRET-435: adds custom webhook for Voleo and MailChimp"  
2e9a1af HEAD@{2}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
9d09113 HEAD@{3}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
990e791 HEAD@{4}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
2ee5f95 HEAD@{5}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
d4d7652 HEAD@{6}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
9adf584 HEAD@{7}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
271de52 HEAD@{8}: commit (amend): PRET-435: adds custom webhook for Voleo and MailChimp  
33b4cdd HEAD@{9}: rebase finished: returning to refs/heads/PRET-435
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (PRET-435)]$ git status  
On branch PRET-435  
Your branch is ahead of 'origin/master' by 1 commit.  
(use "git push" to publish your local commits)  
  
Changes to be committed:  
(use "git reset HEAD <file>..." to unstage)  
  
    modified:   pretio/config/env.py  
    modified:   pretio/database/__init__.py  
    modified:   pretio/database/webhooks/__init__.py  
    deleted:    pretio/database/webhooks/voleo_mailchimp.py  
    modified:   pretio/database/webhooks/webhook.py  
    deleted:    test/database/webhooks/voleo_mailchimp_test.py
```



- common shortcut to undo the last HEAD{0} (Often known as 'undo last commit')
- the - - soft "leaves all your changed files" as seen in 'git status'
- using —hard resets the index and working tree, any changes to tracked files are discarded.

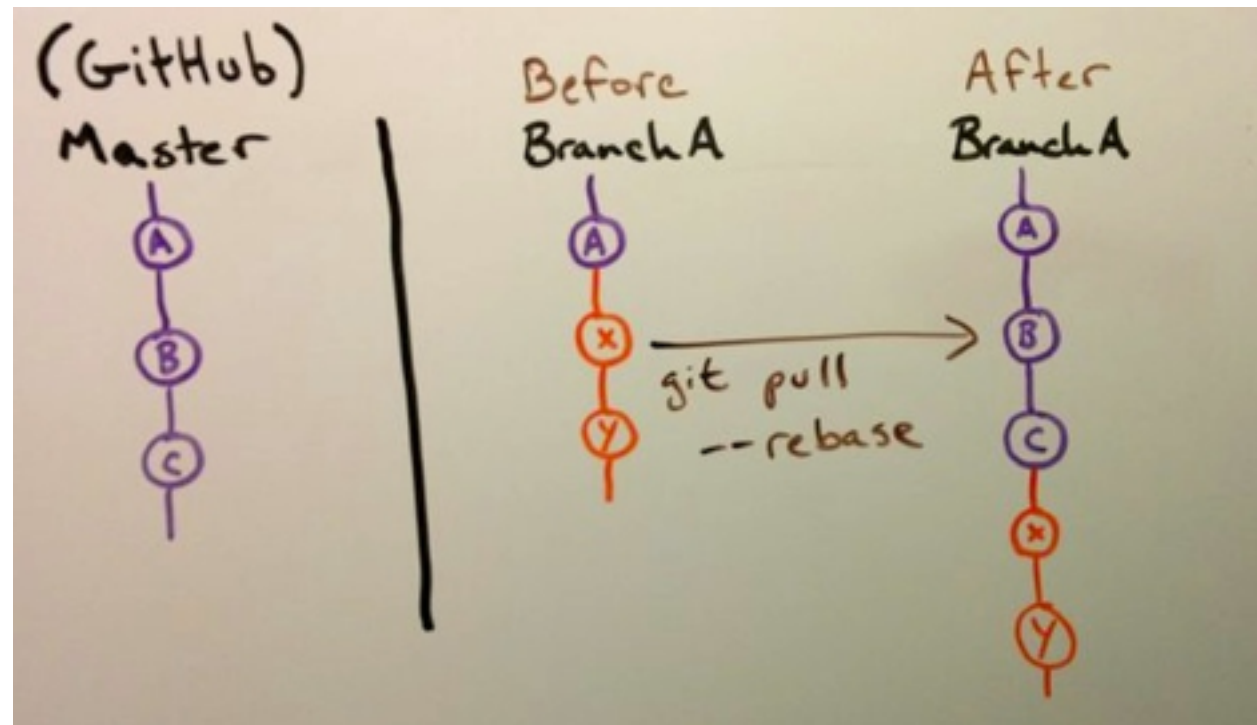


## git rebase vs git merge

- Sometimes folks are... passionate about which is “better”
- Just two different tools for two different jobs



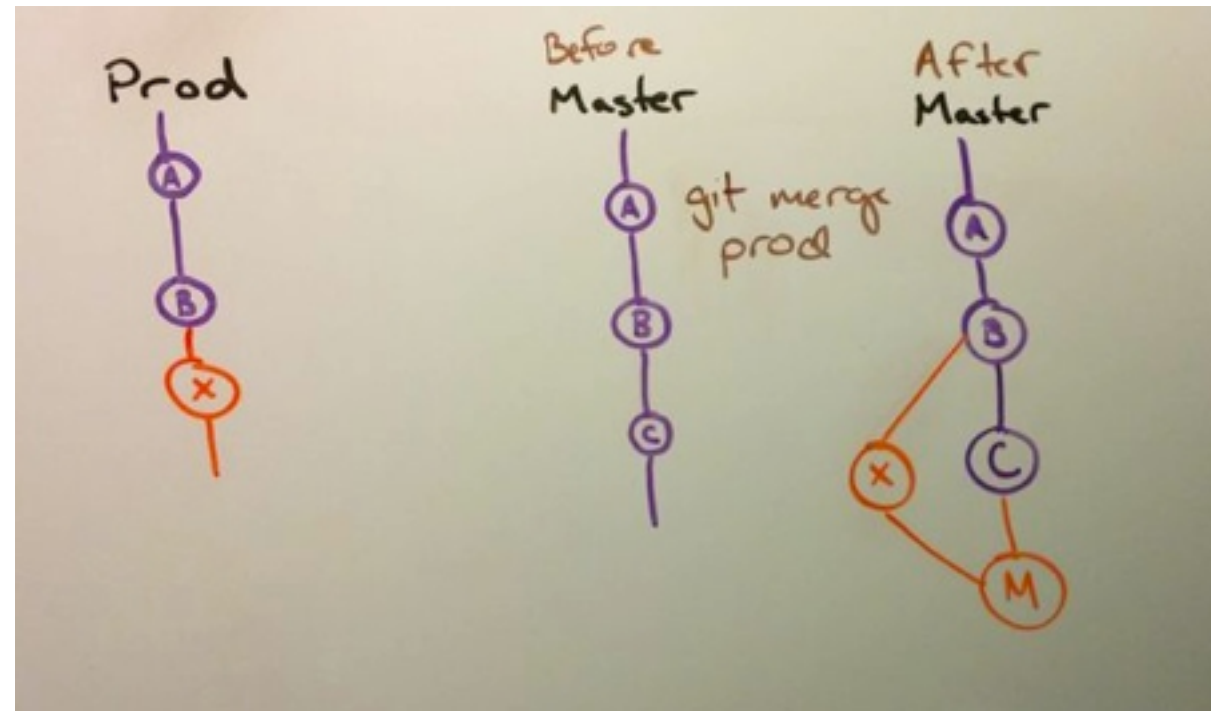
## git rebase



- BranchA is tracking origin/master
- great for pulling down changes from your origin/upstream
- puts your current work on top of any changes that have been made since you ran checkout



## git merge



- 'M' is a new commit indicating the 'merge'
- this method keeps track of the time of 'checkout'
- Note: both merge and rebase need to handle code conflicts!



## Bonus Log Command to Visualize Merges

```
> git log --oneline --decorate --graph
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (branch_b)]$ git log --oneline --decorate --graph
* 8334bed (HEAD, branch_b) Merge branch 'branch_a' into branch_b
|\
| * 7dc8036 (branch_a) a2 on a
| * 6af3129 a1 on a
| * 129ff9f b2 on b
| * dc027e3 b1 on b'
|/
* 706036a (origin/production, origin/master, origin/HEAD, master) Merge pull request #449 f
|\
| * c710f5e Disable experimental celery errors slack integration
| * 7f9c6c6 Run funnel sumamry validation in the correct queue
|/
* c7d581d Merge pull request #446 from Pretio/PRET-396_funnel_summary
|\
```



- this git log command helps illustrate the branching in the drawings
- note this is on a branch\_b that merged in changes from a branch\_a

## git stash

- Take unstaged changes (i.e. what you see when you type *git status* and before you run *git commit*) and... hides them. Somewhere.
- A stack of stashed things accessible from any branch on your local, use it to experiment, put things away temporarily etc.



## git stash

```
> git stash          //stash unstaged changes
> git stash list     //show everything stashed
> git stash pop      //un-stash the top item
```

```
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git stash
Saved working directory and index state WIP on master: 706036a Merge pull request #449 from Pretio/PRET-396_funnel_summary
HEAD is now at 706036a Merge pull request #449 from Pretio/PRET-396_funnel_summary
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

nothing to commit, working directory clean
(pretio)[aurora@Auroras-MacBook-Pro pretio (master)]$ git stash list
stash@{0}: WIP on master: 706036a Merge pull request #449 from Pretio/PRET-396_funnel_summary
```



## A safer way to push with force...

```
> git push origin --force  
// force pushing is destructive and kills  
// work that doesn't match your local history
```



- sometimes there are legit reasons to git push —force, but it is a really destructive action that can overwrite history and loose work, and it's dangerous and easy to make a mistake

## A safer way to push with force...

```
> git push origin --force-with-lease

// checks to see if the force-push is
// destructive and alerts you
```

A full writeup of how force with lease works here:

<https://developer.atlassian.com/blog/2015/04/force-with-lease/>



—force with lease is a lot safer (but not 100% fool proof!) and should be your default instead



Got that? Good. Now it's time for the interactive part...

**LETS ALL COMMIT CODE AND  
SEE WHAT HAPPENS!**

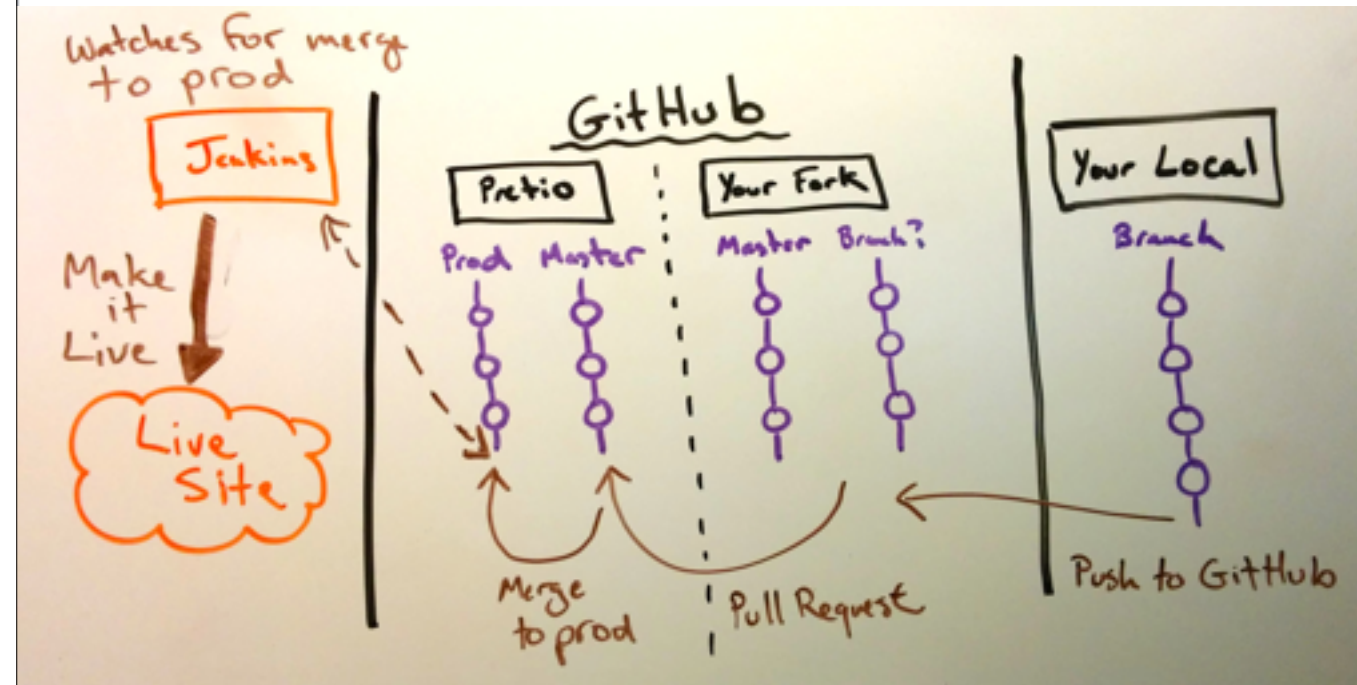


## Interactive Project

- We set up a GitHub repo:
  - [https://github.com/Pretio/startup\\_slam\\_2015\\_website](https://github.com/Pretio/startup_slam_2015_website)
- We have a fake 'JIRA Task Board' with simple tasks to complete.
- You guys get to team up, code up the task, do some code review, and get your code merged.
- Write code solo, in teams, do code review etc. Work together!
- The GitHub repo is also hooked up to Jenkins so it will deploy code LIVE.
- We're not 100% sure what will happen. It'll probably be great.
- Treat the Pretio Team as 'Project Managers'. Feel free to ask for clarification, help etc.



## Visual Aid



# Sources/Reference/Reading

## Thanks To:

<http://chris.beams.io/posts/git-commit/>

// A nice write up on writing good commit messages

<http://nvie.com/posts/a-successful-git-branching-model/>

//The classic document referencing the Git Branching model

<https://github.com/blog/2019-how-to-undo-almost-anything-with-git>

// Notes on handling 'oh no what have I done' situations

<https://developer.atlassian.com/blog/2015/04/force-with-lease/>

//Write up on how force-with-lease works

## Tools:

<https://www.jetbrains.com/idea/>

//Nice development IDE's for a variety of languages with great git integration

## Further Reading:

<https://scotch.io/bar-talk/git-cheat-sheet>

// Fantastic Git Cheat sheet for reference

<https://codewords.recurse.com/issues/two/git-from-the-inside-out>

// Knowing how Git works will level up your Git Wizardy

<http://git-scm.com/docs>

//The actual documentation on Git commands

<http://git-scm.com/book/en/v2>

//The official Pro Git book for free

[https://en.wikipedia.org/wiki/Comparison\\_of\\_continuous\\_integration\\_software](https://en.wikipedia.org/wiki/Comparison_of_continuous_integration_software)

// Jenkins-like tools and alternatives

